An infinite hierarchy of languages defined by dP systems

Gheorghe Păun, Mario J. Pérez-Jiménez

Keywords:

Membrane computing
dP system
Infinite hierarchy
Simple matrix grammar

A B S T R A C T

Here, we continue the study of the recently introduced dP automata. They are symport/antiport P systems consisting of a number of components, each one accepting a string, and working together in recognizing the concatenation of these separate strings; the overall string is distributed to the dP automaton components in a balanced way, i.e., in equal parts up to one symbol, like in the communication complexity area. The question whether or not the number of components induces an infinite hierarchy of the recognized languages was formulated as an open problem in the literature. We solve here affirmatively this question (by connecting P automata with right linear simple matrix grammars), then we also briefly discuss the relation between the balanced and the non-balanced way of splitting the input string among components; settling this latter problem remains as a research topic. Some other open problems are also formulated.

1. Introduction

In the membrane computing area (the reader is referred to [9,12], and to the domain website [15] for details), there are many classes of computing devices, generating or accepting multisets, numbers or strings. Here we deal with devices which accept strings, namely, based on symport/antiport rules. (Couples of objects are passed simultaneously across a membrane, in the same direction in the case of symport and in opposite directions in the case of antiport). Basically, the objects which are taken from the environment during a halting computation are arranged in a string in the order of "reading" them, and the obtained string is said to be accepted by the system. This idea was first explored in [3] (the paper was presented during the Workshop on Membrane Computing, Curtea de Argeş, 2002; the respective devices were called P automata) and, almost concomitantly, in [6], in a simplified version. Several papers were devoted to these devices (in particular, characterizations of regular, context-free, and recursively enumerable languages were obtained, and complexity investigations were carried out); we refer to [2] for details, including references.

Although all P systems are distributed computing machinery, the result of a computation – in particular, the string to be accepted by a P automaton – is produced in a single membrane (or as the input of a single membrane), distinguished in advance. In order to solve a problem – in particular, to accept a string – in a distributed way, a class of P systems was introduced in [10], called dP systems. In the general case, such systems consist of a given number of usual P systems, of any type, which can have their separate inputs and communicate from skin to skin membranes by means of antiport rules (like in tissue-like P systems). In this framework, communication complexity issues can be investigated, as in [7]. The case of P automata (based on symport/antiport rules) was considered in some details – and this leads to the notion of *dP automata*. These devices were further investigated in [5,11], by comparing their power with that of usual P automata and with families

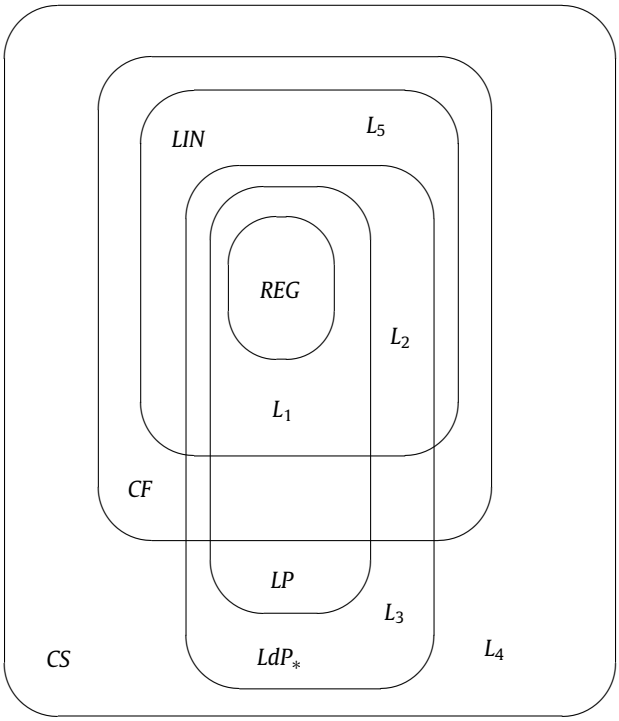

Fig. 1. The place of the families *LP* and *LdP* in Chomsky hierarchy.

of languages in the Chomsky hierarchy. As expected, due to the distribution (and synchronization), dP automata are strictly more powerful than P automata, but the family of languages recognized by them is strictly included in the family of context-sensitive languages. Also expected is the fact that each regular languages can be recognized by a P automaton – see precise details in Fig. 1. (Note that we compare here non-extended P and dP automata, hence without a distinguished terminal alphabet; in the extended case, the P automata are known to be computationally complete.)

A problem left open in [11] asks whether or not the number of components of a dP automaton induces an infinite hierarchy of accepted languages – and this problem is settled here affirmatively. The proof uses a natural connection between dP automata with certain properties and right linear simple matrix grammars, an "old" notion in formal language theory, [8,4]. In this context we provide a simplified proof of the fact that each regular language can be recognized by a P automaton. The possibility to extend such a result/construction to right linear simple matrix grammars and dP automata is formulated as a conjecture.

We also briefly discuss the relationship between the balanced and the non-balanced distribution of a string among the components of a dP automaton, conjecturing that the non-balanced case is more powerful – but the complete solution remains as a task for further research.

2. dP automata

The reader is assumed to have some familiarity with basics of membrane computing, e.g., from [9,12], and of formal language theory, e.g., from [4,13,14], but we recall below all necessary notions.

In what follows, V^* is the free monoid generated by the alphabet V, λ is the empty word, $V^+ = V^* - \{\lambda\}$, $|x|$ denotes the length of the string $x \in V^*$, and $mi(x)$ is the mirror image of $x \in V^*$. *REG*, *LIN*, *CF*, *CS*, *RE* denote the families of regular, linear, context-free, context-sensitive, and recursively enumerable languages, respectively. As usual in membrane computing, the multisets over an alphabet V are represented by strings in V^*; a string and all its permutations correspond to the same multiset, with the number of occurrences of a symbol in a string representing the multiplicity of that object in the multiset. (We work here only with multisets of finite multiplicity.) The terms "symbol" and "object" are used interchangeably, all objects are here represented by symbols.

A *dP automaton* (of degree $n \geq 1$) is a construct

$$\Delta = (O, E, \Pi_1, \ldots, \Pi_n, R),$$

where:

(1) O is an alphabet (of objects);
(2) $E \subseteq O$ (the objects available in arbitrarily many copies in the environment);

(3) $\Pi_i = (O, \mu_i, w_{i,1}, \ldots, w_{i,k_i}, E, R_{i,1}, \ldots, R_{i,k_i})$ is a symport/antiport P system of degree k_i (O is the alphabet of objects, μ_i is a membrane structure of degree k_i, $w_{i,1}, \ldots, w_{i,k_i}$ are the multisets of objects present in the membranes of μ_i in the beginning of the computation, E is the alphabet of objects present – in arbitrarily many copies – in the environment, and $R_{i,1}, \ldots, R_{i,k_i}$ are finite sets of symport/antiport rules associated with the membranes of μ_i; the symport rules are of the form (u, in), (u, out), where $u \in O^+$, and the antiport rules are of the form $(u, out; v, in)$, where $u, v \in O^+$), with the skin membrane labeled with $(i, 1) = s_i$, for all $i = 1, 2, \ldots, n$;

(4) R is a finite set of rules of the form $(s_i, u/v, s_j)$, where $1 \leq i, j \leq n$, $i \neq j$, and $u, v \in O^*$, $uv \neq \lambda$.

The systems $\Pi_1, \ldots, \Pi_n$ are called *components* of Δ and the rules in R are called *communication rules*. For a rule $(s_i, u/v, s_j)$, $|uv|$ is the *weight* of this rule.

Using a rule (u, in), (u, out) associated with a membrane i means to bring in the membrane, respectively out of it the multiset u; using a rule $(u, out; v, in)$ associated with a membrane i means to send out of the membrane the objects of multiset u and, simultaneously, to bring in the membrane, from the region surrounding membrane i, the objects of multiset v. A communication rule $(s_i, u/v, s_j)$ moves the objects of u from the skin region of component Π_i to the skin region of component Π_j, simultaneously with moving the objects in the multiset v in the opposite direction.

Each component Π_i can take an input, work on it by using the rules in sets $R_{i,1}, \ldots, R_{i,k_i}$, and communicate with other components. The communication is done by means of rules in R, but, because the environment is common, the components can also communicate, in two steps, through the environment. In the constructions involved in the proofs of the results given below this latter possibility is systematically avoided.

A halting computation with respect to Δ accepts the string $x = x_1 x_2 \ldots x_n$ over O if the components $\Pi_1, \ldots, \Pi_n$, starting from their initial configurations, using the symport/antiport rules as well as the inter-components communication rules, in the non-deterministic maximally parallel way (at each step, one uses a nondeterministically chosen applicable multiset of rules which is maximal in the sense of inclusion), bring from the environment, symbol by symbol, the substrings $x_1, \ldots, x_n$, respectively, and eventually halts.

The dP automata are synchronized devices, a universal clock exists for all components, marking the time in the same way for the whole dP automaton.

The string to be accepted can be distributed to the dP system components in either the *balanced way* or in the arbitrary way. Like in communication complexity area, [7], balanced means to have the parts equal in length modulo one symbol. Formally, for a dP automaton Δ of degree n we define the language $L(\Delta)$, of all strings $x \in O^*$ such that we can write $x = x_1 x_2 \ldots x_n$, with $||x_i| - |x_j|| \leq 1$ for all $1 \leq i, j \leq n$, each component Π_i of Δ takes as input the string x_i, $1 \leq i \leq n$, and the computation halts. (The restriction to have the string distributed in a balanced way is here a *property* of the system, not an external condition.) If this restriction is not imposed, hence any decomposition of the string x can be considered, then a superlanguage of $L(\Delta)$ is obtained, which we denote by $L'(\Delta)$. (Note that we do not take here into account also the communication complexity of accepting a string, as done in [1,10].)

We denote by LdP_n, $L'dP_n$ the families of languages $L(\Delta)$, $L'(\Delta)$, respectively, for Δ of degree at most n. A dP automaton of degree 1 is a usual P automaton – of a non-extended type: all symbols are introduced in the accepted string. If a terminal set of objects is considered, then we obtain an extended P automaton (formally, we have a device $\Pi = (O, T, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m)$, with $T \subseteq O$, working as a usual P automaton and considering only the symbols from T in the accepted strings and ignoring those from $O - T$). We denote by LP the family of languages recognized by non-extended P automata (hence $LP = LdP_1$) and by ELP the family of languages recognized by extended P automata. (Note that we ignore the weight of symport and antiport rules, but these parameters, usual when investigating symport/antiport P systems, can be considered also here.) If the subscript n in LdP_n or $L'dP_n$ is arbitrary, then we replace it by $*$.

A terminal alphabet can be considered also for dP automata, but this is not of much interest: $ELdP_1 = ELP$, which is known to equal RE.

Specific dP automata will be given in the proofs below, hence we do not provide examples of such devices here.

3. The power of P and dP automata

We recall from [5,11] the diagram in Fig. 1, indicating the relationships between families of languages accepted by (non-extended) P and dP automata and families in the Chomsky hierarchy. The languages indicated in the diagram are as follows (f is the morphism defined by $f(a) = a', f(b) = b'$):

$$L_1 = \{(a^2c)^s(b^2d)^s \mid s \geq 1\},$$
$$L_2 = \{(ab)^s(ac)^s \mid s \geq 1\},$$
$$L_3 = \{wf(w) \mid w \in \{a, b\}^*\},$$
$$L_4 = \{(wf(w))^s \mid w \in \{a, b\}^+, s \geq 2\},$$
$$L_5 = \{w\, mi(w) \mid w \in \{a, b\}^*\},$$

Note that the fact that $L_5 \notin LdP_*$ is only *a conjecture*, but the place of other languages in the diagram is proved in [5,11].

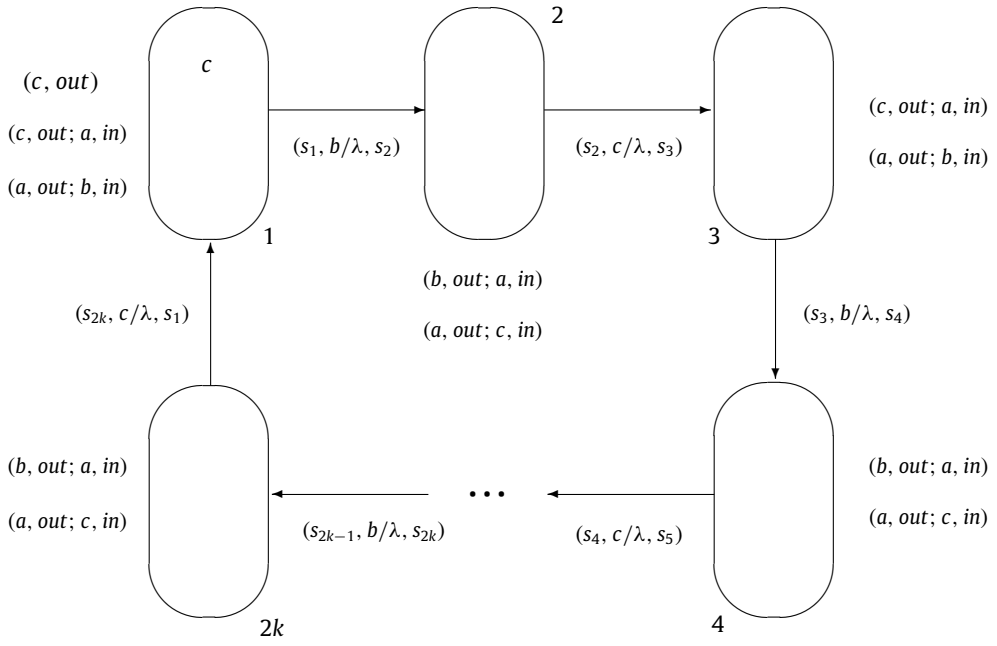

Fig. 2. The dP system in the proof of Lemma 4.1.

4. The hierarchy of families LdP_n

By definition, we have the inclusion $LdP_n \subseteq LdP_{n+1}$ for all $n \geq 1$, and $LdP_1 = LP \subset LdP_2$, but it is not known whether also the other inclusions are proper. We prove here that this is the case, by using the following sequence of languages (and variations of them):

$$L_k = \{((ab)^m(ac)^m)^k \mid m \geq 0\}, \quad k \geq 1.$$

Lemma 4.1. *$L_k \in LdP_{2k}$ for all $k \geq 1$.*

Proof. We consider the following dP automaton (also given in a graphical form in Fig. 2):

$$\begin{aligned}
\Delta &= (O, E, \Pi_1, \ldots, \Pi_{2k}, R), \text{ where :}\\
O &= E = \{a, b, c\},\\
\Pi_1 &= (O, [\ ]_1, c, E, \{(c, out), (c, out; a, in), (a, out; b, in)\},\\
\Pi_i &= (O, [\ ]_i, \lambda, E, \{(b, out; a, in), (a, out; c, in)\}, \quad i = 2, 4, \ldots, 2k,\\
\Pi_i &= (O, [\ ]_i, \lambda, E, \{(c, out; a, in), (a, out; b, in)\}, \quad i = 3, 5, \ldots, 2k-1,\\
R &= \{(s_i, b/\lambda, s_{i+1}) \mid i = 1, 3, \ldots, 2k-1\}\\
&\quad \cup \{(s_i, c/\lambda, s_{i+1}) \mid i = 2, 4, \ldots, 2k-2\} \cup \{(s_{2k}, c/\lambda, s_1)\}.
\end{aligned}$$

We start with a unique object inside the system, c in component Π_1, and always we have only one object in the system, circulating along the cycle $\Pi_1, \Pi_2, \ldots, \Pi_{2k}, \Pi_1$. When visited by this object, each component takes two objects from the environment, in two consecutive steps; the second object (b or c) cannot go to the environment, hence it should move to the next component, thus the substrings recognized by each component grow synchronously. When the object c returns to the first component, it can start a new cycle or it can exit the system, by the rule (c, out), and the computation halts. Therefore, $L(\Delta) = L_k$, and the proof is completed. □

Appending a block $(ad)^m$ to the strings in L_k we can get a language L'_k which will belong to LdP_{2k+1}, thus covering also the case of dP systems of an odd degree; here and in what follows, the necessary changes in the proofs are left to the reader, and we continue by considering only the languages L_k.

The languages L_k have a property which makes them easy to handle in terms of dP automata: no two adjacent symbols can be swapped without leaving the language. Because of its usefulness, we give a name to this property: a language $L \subseteq V^*$ is said to be *frozen* if there is no string $xaby \in L$, for some $a, b \in V$, $x, y \in V^*$, such that $xbay \in L$. (In particular, the two symbols can be identical, hence no string in L contains a substring a^2, for $a \in V$.)

Now, if a language $L(\Delta)$ is frozen, the components of the dP automaton Δ can never bring two or more objects from the environment at the same time (then either of the permutations of these symbols is a substring of the accepted string). Therefore, the rules by which the automaton can bring objects inside can be of one of the two forms: (a, in), $(u, out; a, in)$, for some object a and multiset u. However, if $a \in E$ (i.e., a is available in the environment in arbitrarily many copies), then a rule (a, in) is not allowed, as the system would be flooded with infinitely many objects. That is, such rules can be used only for symbols present in the initial configuration of Δ and not in E. In turn, rules $(u, out; a, in)$ do not increase the number of objects inside the system. Consequently, the dP automaton never contains more objects than in the initial configuration, which means that the number of configurations reachable from the initial configuration is finite. We say that the dP automaton itself is *finite*.

This makes possible the simulation of a dP automaton (generating a frozen language) by means of a *right linear simple matrix grammar* in the sense of [8]—see also [4].

Such a grammar of degree $n \geq 1$ is a construct of the form $G = (N_1, \ldots, N_n, T, S, M)$, where $N_1, N_2, \ldots, N_n, T$ are pairwise disjoint alphabets (we denote by N the union of $N_1, \ldots, N_n$), $S \notin T \cup N$, and M contains matrices (of context-free rules) of the following forms:

(i) $(S \to x), x \in T^*$,
(ii) $(S \to A_1 A_2 \ldots A_n),\ A_i \in N_i, 1 \leq i \leq n$,
(iii) $(A_1 \to x_1 B_1, \ldots, A_n \to x_n B_n),\ A_i, B_i \in N_i, x_i \in T^*, 1 \leq i \leq n$,
(iv) $(A_1 \to x_1, \ldots, A_n \to x_n),\ A_i \in N_i, x_i \in T^*, 1 \leq i \leq n$.

A derivation starting with a matrix of type (ii) continues with an arbitrary numbers of steps which use matrices of type (iii) and ends by applying a matrix of type (iv).

We denote by $L(G)$ the language generated in this way by G and by RSM_n the family of languages $L(G)$ for right linear simple matrix grammars G of degree at most n, for $n \geq 1$. The union of all these families is denoted by RSM_*.

Clearly, a normal form can be easily found for these grammars: in matrices of type (iii) we can ask to have $x_i \in T \cup \{\lambda\}$, $1 \leq i \leq n$, and in matrices of type (iv) to have $x_i = \lambda$ for all $1 \leq i \leq n$.

The similarity of producing a string in a finite dP system and in a right linear simple matrix grammar is apparent, and this makes expected the following result (and construction).

Lemma 4.2. *If $L \in LdP_k$ and L is frozen, then $L \in RSM_k$, for all $k \geq 1$.*

Proof. Let Δ be a dP automaton of degree k (with the set of objects O) recognizing a frozen language; as observed above, the dP automaton is then finite. Let $\sigma_0, \sigma_1, \ldots, \sigma_p$ be the set of all configurations of Δ which can be reached from the initial configuration, σ_0. We construct the following right linear simple matrix grammar:

$$
\begin{aligned}
G &= (N_1, \ldots, N_k, O, S, M), \text{ with} \\
N_i &= \{(\sigma_j)_i \mid 0 \leq j \leq p\}, \quad i = 1, 2, \ldots, k, \\
M &= \{(S \to (\sigma_0)_1 (\sigma_0)_2 \ldots (\sigma_0)_k)\} \\
&\quad \cup \{(\sigma_i)_1 \to \alpha_1 (\sigma_j)_1, \ldots, (\sigma_i)_k \to \alpha_k (\sigma_j)_k) \mid \\
&\quad \text{from configuration } \sigma_i \text{ the dP automaton } \Delta \text{ can pass to} \\
&\quad \text{the configuration } \sigma_j \text{ by a transition, taking from the} \\
&\quad \text{environment the objects } \alpha_1, \ldots, \alpha_k \text{ by its } k \text{ components, where} \\
&\quad \alpha_s \in O \cup \{\lambda\}, 1 \leq s \leq k\} \\
&\quad \cup \{(\sigma_h)_1 \to \lambda, \ldots, (\sigma_h)_k \to \lambda) \mid \sigma_h \text{ is a halting configuration}\}.
\end{aligned}
$$

Note that all nonterminals in the rules of a matrix contain the same "core information", namely the current configuration of the system, hence the complete control of the system working is obtained in this way. The equality $L(\Delta) = L(G)$ is obvious. □

The previous result (and construction) cannot be extended to arbitrary languages in LdP_*: remember that $ELP = RE$, while each language in ELP is obtained from a language in $LP = LdP_1$ by means of an erasing morphism which removes the objects which we do not want to keep as "terminal". However, the family RSM_* is closed under arbitrary morphisms (Theorem 1.5.6 in [4]). If we had $LdP_* \subseteq RSM_*$, then $RSM_* = RE$, which is not true (RSM_* is placed in the Chomsky hierarchy in a similar position as LdP_* in Fig. 1: it includes REG, is included in CS, and it is incomparable with LIN and CF). Thus, we have the next result:

Corollary 4.1. $LdP_* - RSM_* \neq \emptyset$.

Actually, RSM_* can be replaced here with the much larger family $CFSM_*$, of languages generated by context-free simple matrix grammars, which is also closed under arbitrary morphisms and is strictly included in CS.

It would be worth recalling here a technical result from [4], Lemma 1.5.4, which basically says that any language in RSM_k can be "projected" on the k "components" of a right linear simple matrix grammar such that the k obtained languages are regular (we state here the version dealing with this particular case, but it also holds, in adequate versions, for context-free and linear simple matrix grammars).

Lemma 1.5.4 ([4]). *For each language $L \in RSM_n$ there are n regular languages $L_1, L_2, \ldots, L_n$ such that:*

(i) $L \subseteq L_1 L_2 \ldots L_n$,
(ii) *for each i, $1 \leq i \leq n$, and for each $x_i \in L_i$, there are $x_j \in L_j$, $1 \leq j \leq n$, $i \neq j$, such that $x_1 x_2 \ldots x_n \in L$.*

A property as in this lemma is also visible from the construction of the grammar G in the previous proof, and this leads to the following result.

Corollary 4.2. *$L_k \notin LdP_{2k-1}$ for all $k \geq 1$.*

Proof. Start from the language L_k, generated by a dP automaton Δ (of degree $2k$, as in Lemma 4.1), and perform the construction of a right linear simple matrix grammar G as in the proof of Lemma 4.2. Then, remove all objects α_i, for $i \geq 2$, from all matrices of G. All rules different from those on the first position in each matrix are now useless, their information is also contained in the first rule. By removing these superfluous rules, we get a right linear grammar G', generating the language of strings accepted by the first component of Δ. Consequently, this language is regular.

However, distributing in a balanced way a string $((ab)^m(ac)^m))^k$ into $2k-1$ parts which are equal modulo one symbol, we get a string $(ab)^m(ac)^{m/(2k-1)}$ in the first component.

Indeed, the string should be distributed in parts of length

$$\frac{|((ab)^m(ac)^m)^k|}{2k-1} = \frac{4mk}{2k-1},$$

hence, besides $(ab)^m$ (of length $2m$), the first component must also read a string $(ac)^x$ (plus or minus one symbol) such that

$$2m + 2x = \frac{4mk}{2k-1},$$

which, by a simple calculation, gives $x = m/(2k-1)$.

The language of all such strings, for $m \geq 0$, is not regular, a contradiction which prevents the existence of a dP automaton of degree $2k-1$ which generates L_k. □

We have obtained the strict inclusion $LdP_{2k-1} \subset LdP_{2k}$ for all $k \geq 1$. As suggested before, by a simple modification of the languages L_k we can get also the strictness of inclusions $LdP_{2k} \subset LdP_{2k+1}$, $k \geq 1$, hence we have the main result of this paper:

Theorem 4.1. *LdP_k, $k \geq 1$, is an infinite hierarchy, all inclusions $LdP_k \subset LdP_{k+1}$, $k \geq 1$, being proper.*

A counterpart to the result in Lemma 4.2, i.e., the inclusion $RSM_* \subseteq LdP_*$, is rather plausible. We *conjecture* that this inclusion holds true (the normal form mentioned before Lemma 4.2 could be useful in proving this). For instance, this conjecture is supported by the fact that $REG \subset LP$, as proved in [5]. Passing from REG to RSM_n looks like passing from a usual P automaton to a dP automaton of degree n, so it might be possible to extend the construction in [5] to RSM_n. That is why we provide here a new proof of the inclusion $REG \subseteq LP$, significantly simpler than that in [5].

Theorem 4.2. *$REG \subseteq LP$.*

Proof. Let us consider a finite automaton A written as in [14], $A = (V, K, s_0, s_f, P)$, where V is the alphabet, K is the set of states, s_0 is the initial state, s_f is the final state, and P is a finite set of transitions, given as rewriting rules of the form $s_i a \to s_j$, for $s_i, s_j \in K$, $a \in V$. Without loss of generality, we may assume that there is no rule of the form $s_f a \to s_j$ in P (if there is such a rule, then we consider a new final state, s'_f, and for each rule $s_i a \to s_f$ we also introduce in P the rule $s_i a \to s'_f$; this change leads to an equivalent finite automaton).

Let K' be the set $\{s'_i \mid s_i \in K\}$. For a finite set X, we identify by X also the multiset consisting of one copy of each element in X.

We construct now the following P automaton—its initial configuration is shown in Fig. 3.

$$\begin{aligned}
\Pi &= (O, \mu, w_1, w_2, w_3, E, R_1, R_2, R_3),\\
O &= V \cup K \cup K' \cup \{d, \#\},\\
\mu &= [\,[\;\;]_2[\;\;]_3\,]_1,\\
w_1 &= d, \qquad w_2 = V \cup K \cup K', \qquad w_3 = s_0\#,\\
E &= V,\\
R_1 &= \{(d, out, a, in) \mid a \in V\}\\
&\quad \cup \{(a, out; b, in) \mid a, b \in V\},\\
R_2 &= \{(s'_j a, out; s_i a, in),\ (s_j, out; s'_j, in) \mid s_i a \to s_j \in P\}\\
&\quad \cup \{(s_i a, in) \mid s_i a \to s_f \in P\}\\
&\quad \cup \{(\#, in),\ (\#, out)\},\\
R_3 &= \{(s_0, out)\} \cup \{(\#, out; s_i, in) \mid s_i \in K\}.
\end{aligned}$$

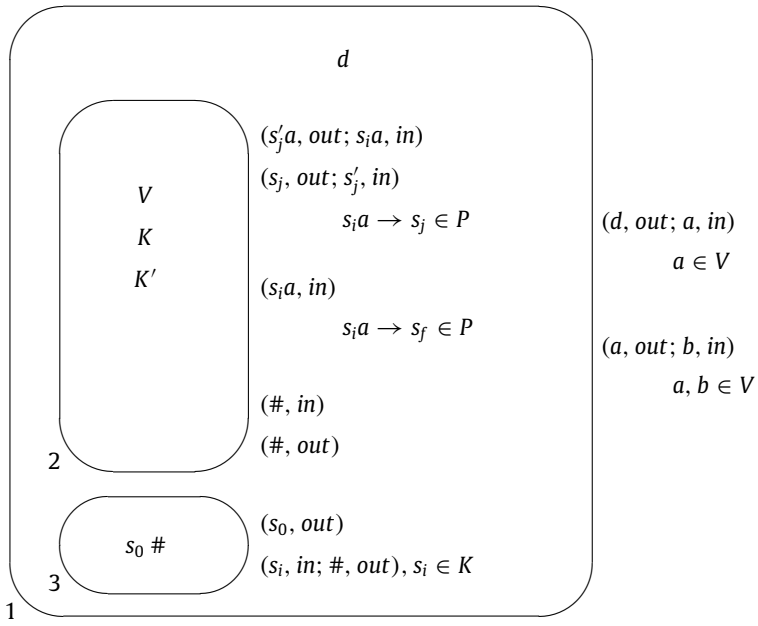

Fig. 3. Simulating a finite automaton by a P automaton.

The automaton works as follows. In the first step, the state-object s_0 is released from membrane 3, and a rule $(d, out; a, in)$ brings an object $a \in V$ from the environment.

No object $s_i \in K$ can stay unused in the skin membrane, because otherwise it releases the trap object # from membrane 3, which then will oscillate forever across membrane 2. Thus, we cannot use once again a rule from R_1, bringing one further object from the environment, but we have to use a rule $(s'_j a, out; s_i a, in)$ from R_2, the only other rule applicable at this step, and such a rule exists if and only if a rule $s_i a \to s_j$ exists in the finite automaton.

Now, the object a will bring a new symbol $b \in V$ from the environment, simultaneously with exchanging the primed version of s'_j with the unprimed one. We are in a situation as after the first step.

The process continues as long as rules from P can handle the current state present in membrane 1 and the symbol from V brought inside. If this is not the case, the trap object is released and the computation never stops.

If a correct parsing in A is followed, and we reach the final state s_f by a rule $s_i a \to s_f$, then the rule in R_2 to use is $(s_i a, in)$, and the computation halts. The only objects taken from the environment were those of the string recognized by the automaton A, that is, $L(\Pi) = L(A)$. □

5. Balanced versus non-balanced distribution

We expect to have a separation between these cases, with a strict decrease in power in the case of the balanced distribution of strings, but we only settle here this question for dP systems with two components. Namely, we consider the language

$$K_4 = \{a^{n-1}b^{3n} \mid n \geq 2\}.$$

We have $K_4 \in LdP_4$, as proved by the dP automaton in Fig. 4. This system works as follows. The object a in Π_1 and objects b in Π_2, Π_3, Π_4 repeatedly bring copies of a, respectively, b from the environment, thus increasing the accepted string. The process can continue as long as the membrane $1'$ does not expel the four copies of c present in it (they can oscillate across membrane $1''$ for an arbitrary number of steps and then, nondeterministically, can exit membrane $1'$). Object c cannot stay in membrane 1. If two copies of it arrive in the same component Π_2, Π_3, Π_4, then one of it can exit together with b, but the other will release the trap object # and the computation never stops (object # will exit the system and enter back indefinitely by means of rules (#, *out*), (#, *in*) present in each of Π_2, Π_3, Π_4). Therefore, exactly one c has to arrive in each component Π_2, Π_3, Π_4 and the one remaining in Π_1 exits together with a (if it exits together with #, then the computation continues forever). In this way, the computation halts. Note that while c comes to Π_2, Π_3, Π_4, one additional b is brought into these components, hence indeed we recognize the language K_4.

This language is not in the family LdP_2: distributing a string $a^{m-1}b^{3m}$ in a balanced way into two parts (equal modulo at most one symbol) means getting $a^{m-1}b^{m+x}$ with $x \in \{0, 1\}$ in the fist component and b^{2m-x} in the second one. The language

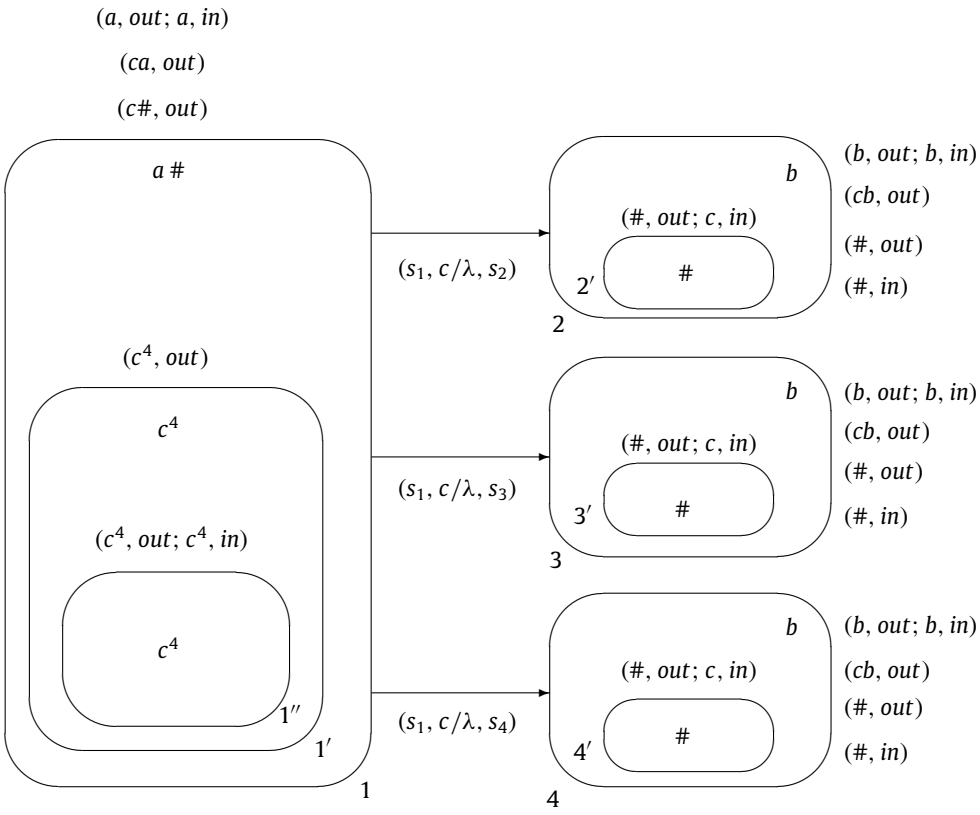

Fig. 4. A dP automaton of degree 4 recognizing the language K_4.

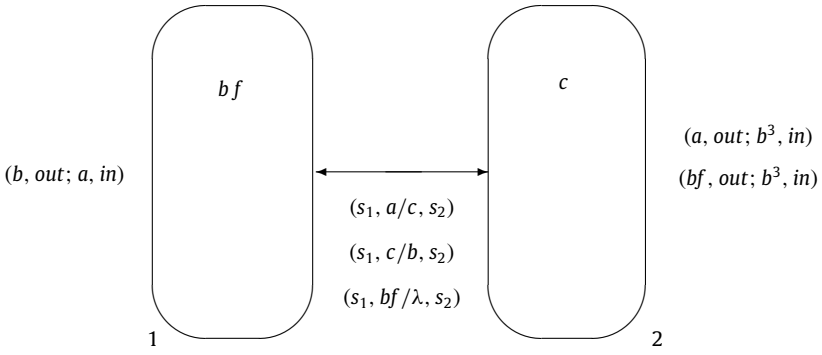

Fig. 5. A dP automaton recognizing in the unbalanced way the language K_4.

of all such strings, $a^{m-1}b^{m+x}$ with $x \in \{0, 1\}$, for $m \geq 0$, is not regular, which contradicts the result established in the proof of Corollary 4.2 showing that the language of strings accepted by the first component of a dP automaton is regular.[1]

However, K_4 is in $L'dP_2$, as proved by the simple dP automaton from Fig. 5. We start by introducing the object a in the first component. This object goes to Π_2 (in exchange of object c) and there brings three copies of b. Two of these copies exit immediately, the third one must go to Π_1, in exchange of c, hence the configuration is restored. The process is iterated until using the communication rule $(s_1, bf/\lambda, s_2)$; bf will introduce one further block b^3 in Π_2 and the computation stops.

[1] The relation $K_4 \notin LdP_2$ was pointed out to us by a referee.

6. Final remarks

A counterpart of Theorem 4.1 is probably true: the language L_k cannot be recognized by a dP automaton with less than $2k$ components (Corollary 4.2), but, symmetrically, it cannot be recognized by a dP automaton with more than $2k$ components either. For instance, if we have $2k + 1$ components, then, because of the balanced distribution of the strings $((ab)^m(ac)^m)^k$, the second component does not accept a regular language (the first component accepts a prefix of $(ab)^m$, but the second one accepts a suffix of $(ab)^m$ concatenated with a prefix of $(ac)^m$, the two strings being of a total length which is imposed by the balanced distribution—and such a string cannot be regular). The details are left to the reader, as well as the task to consider further cases than that of dP automata of degree $2k + 1$. (Are there values $s \geq 2k + 1$ such that $L_k \in LdP_s$?)

The property of a language to be frozen is very "fragile": all standard operations in language theory (except the mirror image) make frozen languages to become non-frozen. In particular, the family of frozen languages form an anti-AFL (it is closed to none of the six AFL operations: union, concatenation, Kleene $+$, morphisms, inverse morphisms, and intersection with regular languages) — which is easy to prove.

Many other problems remain to be investigated with respect to dP automata – several of them can be found in [11], hence we refer the reader to that paper.

Acknowledgements

This work is supported by the Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200. Thanks are due to three anonymous referees who have carefully read the paper.

References

[1] H. Adorna, Gh. Păun, M.J. Pérez-Jiménez, On communication complexity in evolution-communication P systems, in: M.A. Martínez-del-Amor, et al. (Eds.), Proc. 8th Brainstorming Week on Membrane Computing, Sevilla, February 2010, Fenix Editora, 2010, pp. 1–21; Romanian Journal of Information Theory and Applications 13 (2) (2010) 113–130.
[2] E. Csuhaj-Varjú, M. Oswald, G. Vaszil, P automata, Chapter 6 in [12], pp. 144–167.
[3] E. Csuhaj-Varjú, G. Vaszil, P automata or purely communicating accepting P systems, in: Gh. Păun, et al. (Eds.), Membrane Computing. International Workshop, WMC 2002, Curtea de Argeş, Romania, August 2002. Revised Papers, in: LNCS, vol. 2597, Springer, 2003, pp. 219–233.
[4] J. Dassow, Gh. Păun, Regulated Rewriting in Formal Language Theory, Springer, Berlin, 1989.
[5] R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez, On the power of P and dP automata, Annals of Bucharest University. Mathematics-Informatics Series 63 (2009) 5–22.
[6] R. Freund, M. Oswald, A short note on analysing P systems, Bulletin of the EATCS 79 (October) (2002) 231–236.
[7] J. Hromkovic, Communication Complexity and Parallel Computing: The Application of Communication Complexity in Parallel Computing, Springer, Berlin, 1997.
[8] O. Ibarra, Simple matrix grammars, Information and Control 17 (1970) 359–394.
[9] Gh. Păun, Membrane Computing. An Introduction, Springer, Berlin, 2002.
[10] Gh. Păun, M.J. Pérez-Jiménez, Solving problems in a distributed way in membrane computing: dP systems, International Journal of Computers, Communication and Control 5 (2) (2010) 238–252.
[11] Gh. Păun, M.J. Pérez-Jiménez, P and dP automata: a survey, in: C.S. Calude, G. Rozenberg, A. Salomaa (Eds.), Rainbow of Computer Science, in: LNCS, vol. 6570, Springer, Berlin, 2011, pp. 102–115.
[12] Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), Handbook of Membrane Computing, Oxford University Press, 2010.
[13] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 3, Springer, Berlin, 1998.
[14] A. Salomaa, Formal Languages, Academic Press, New York, 1973.
[15] The P Systems Website: http://ppage.psystems.eu.